Computer Systems Architecture

Steven R. Bagley

Introduction

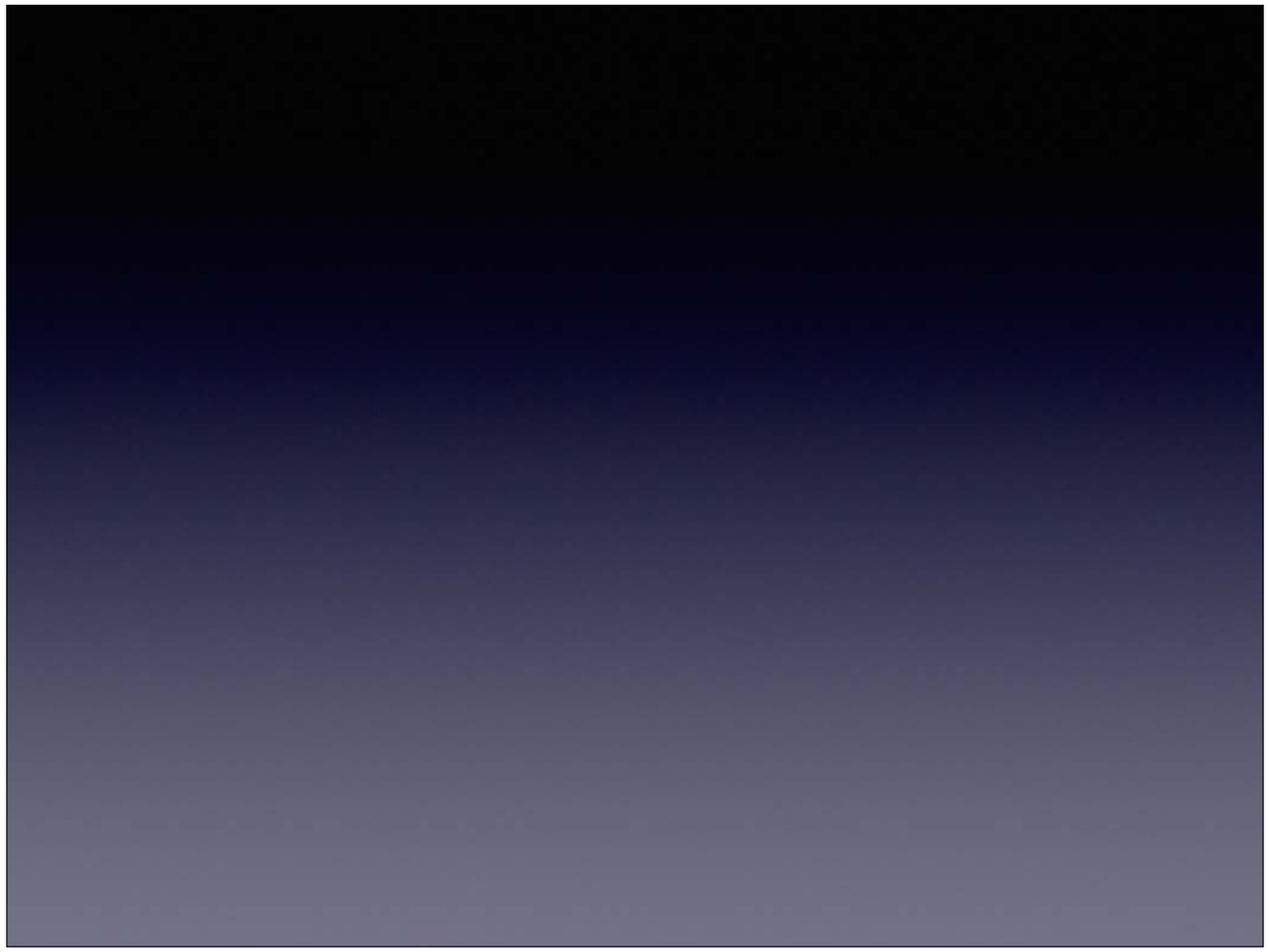
- Programming is about telling the computer what we want it to do
- CSA is about getting ‘under-the-hood’
- Understand how the computer does what we tell it
- How we can build one from the ground up

Introduction

- Overview of the subject
- Going to have to think about topics that branch out of Computer Science
  - Electronics, Maths, Logic, Physics…
- Layers of abstraction tend to shield us for the most part from this
- But occasionally things leak through…

Very much my overview, what I've found interesting and important

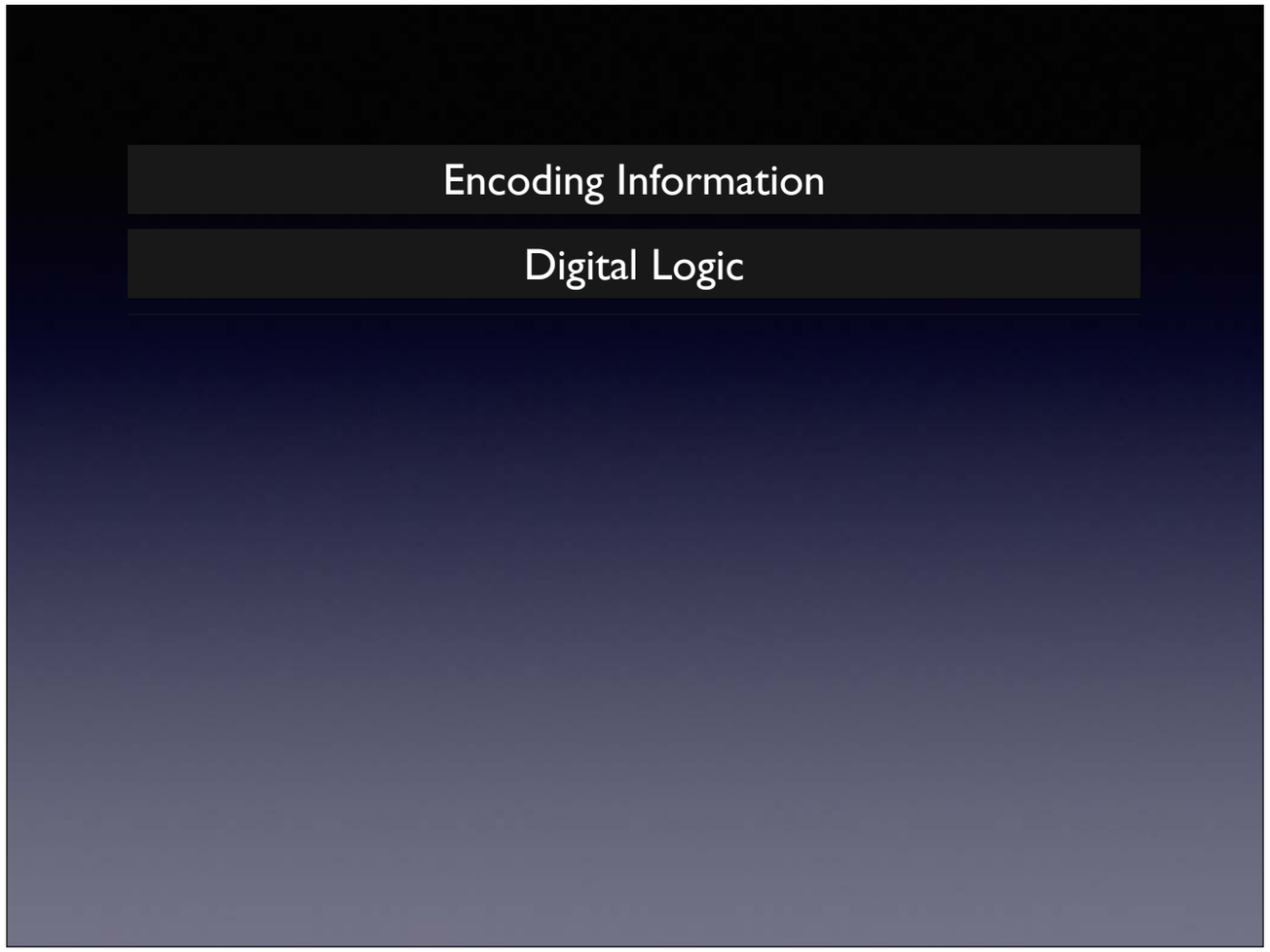
Helpful to know what's going on when we hit the leaky bits of the abstraction



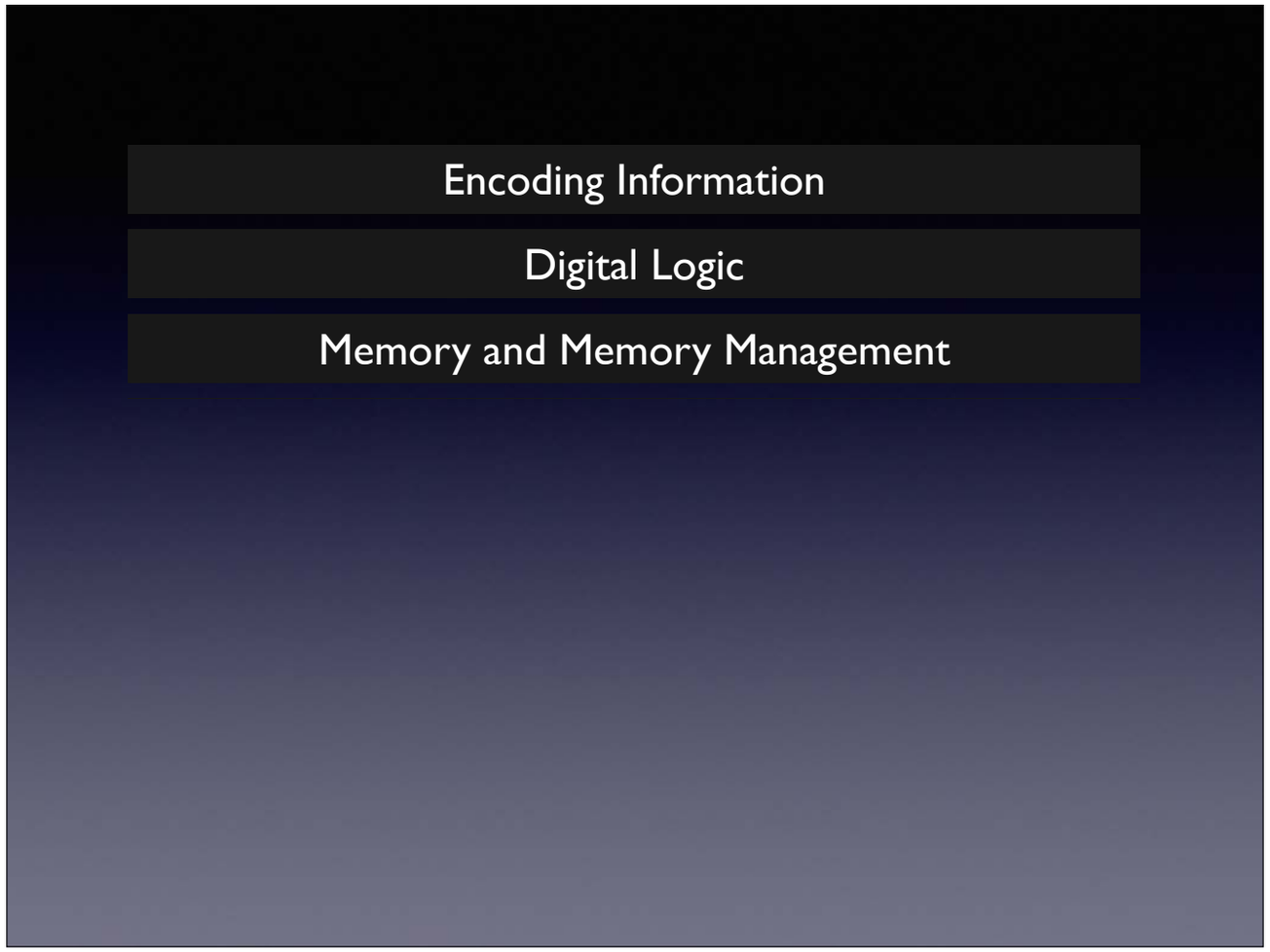
Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course

Encoding Information

Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course

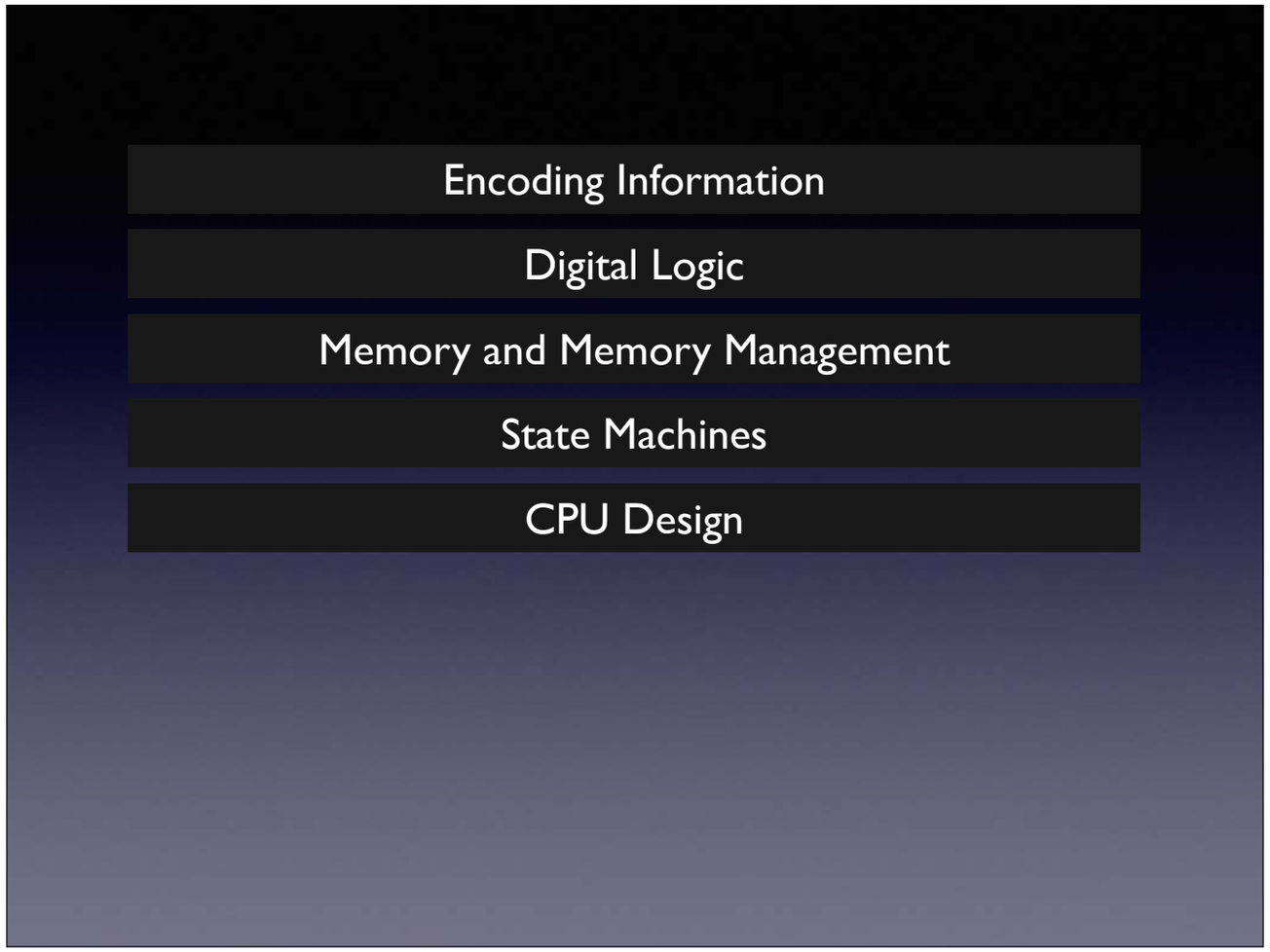
Encoding Information

Digital Logic

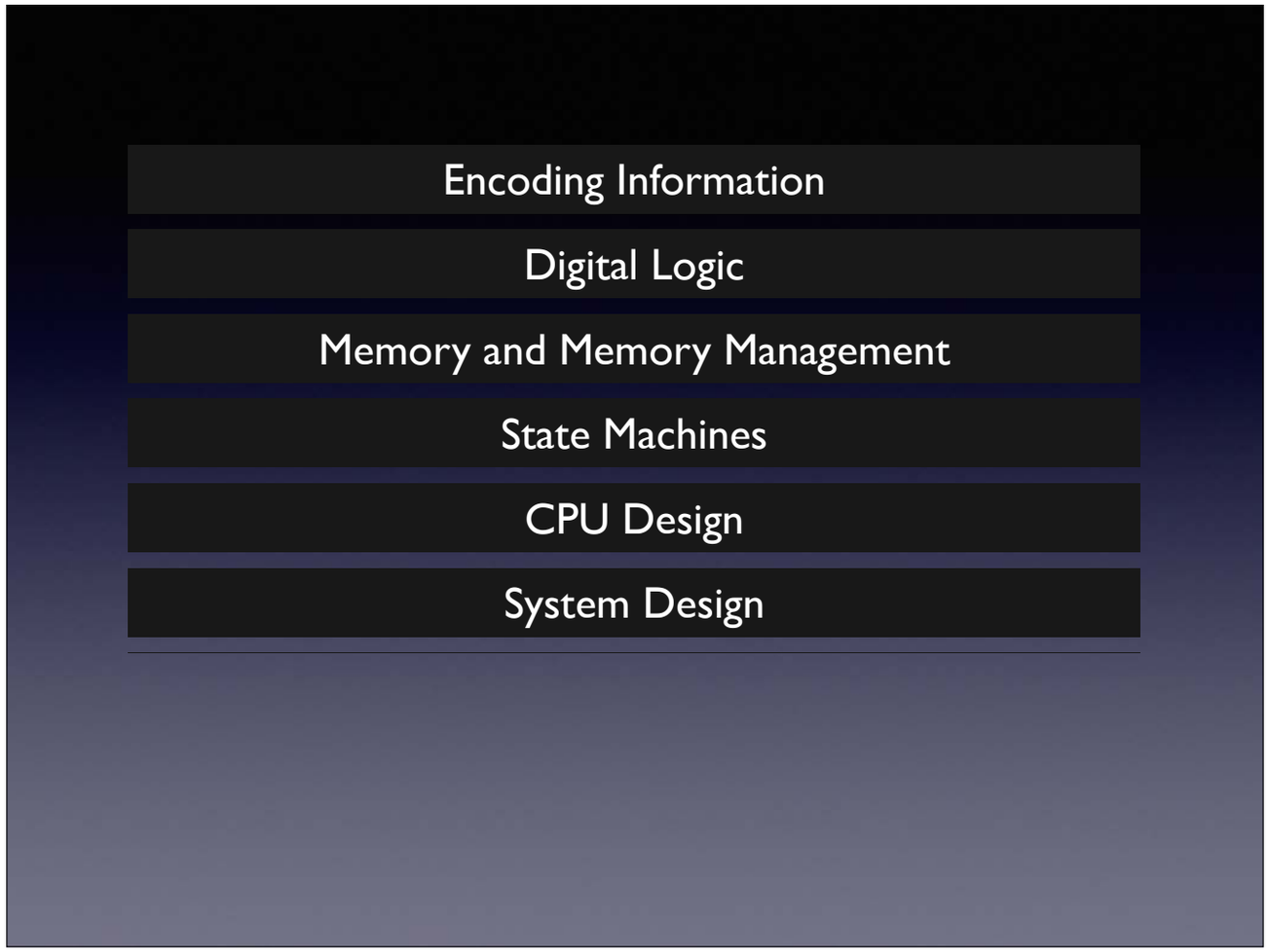
Memory and Memory Management

State Machines

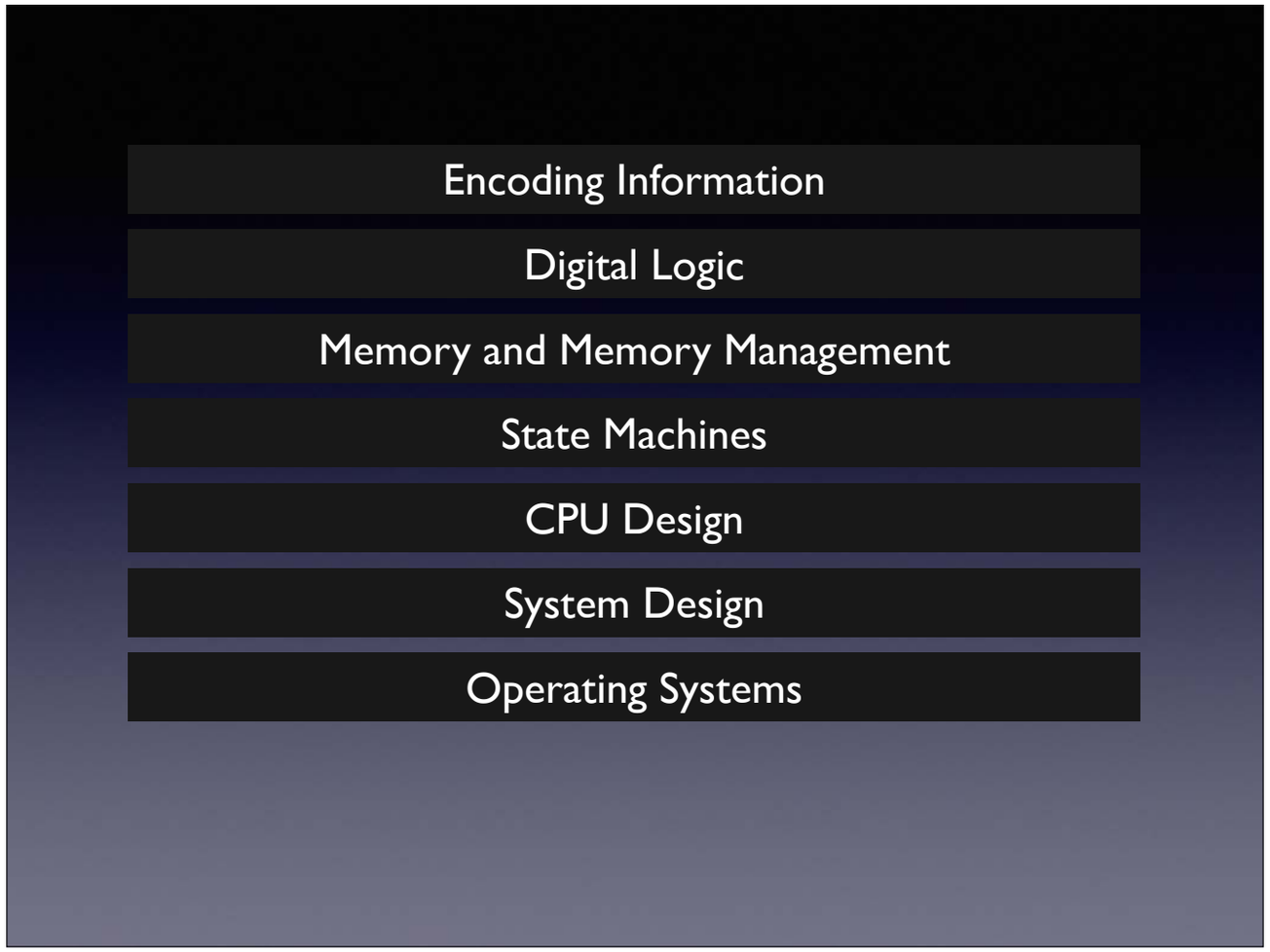
Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



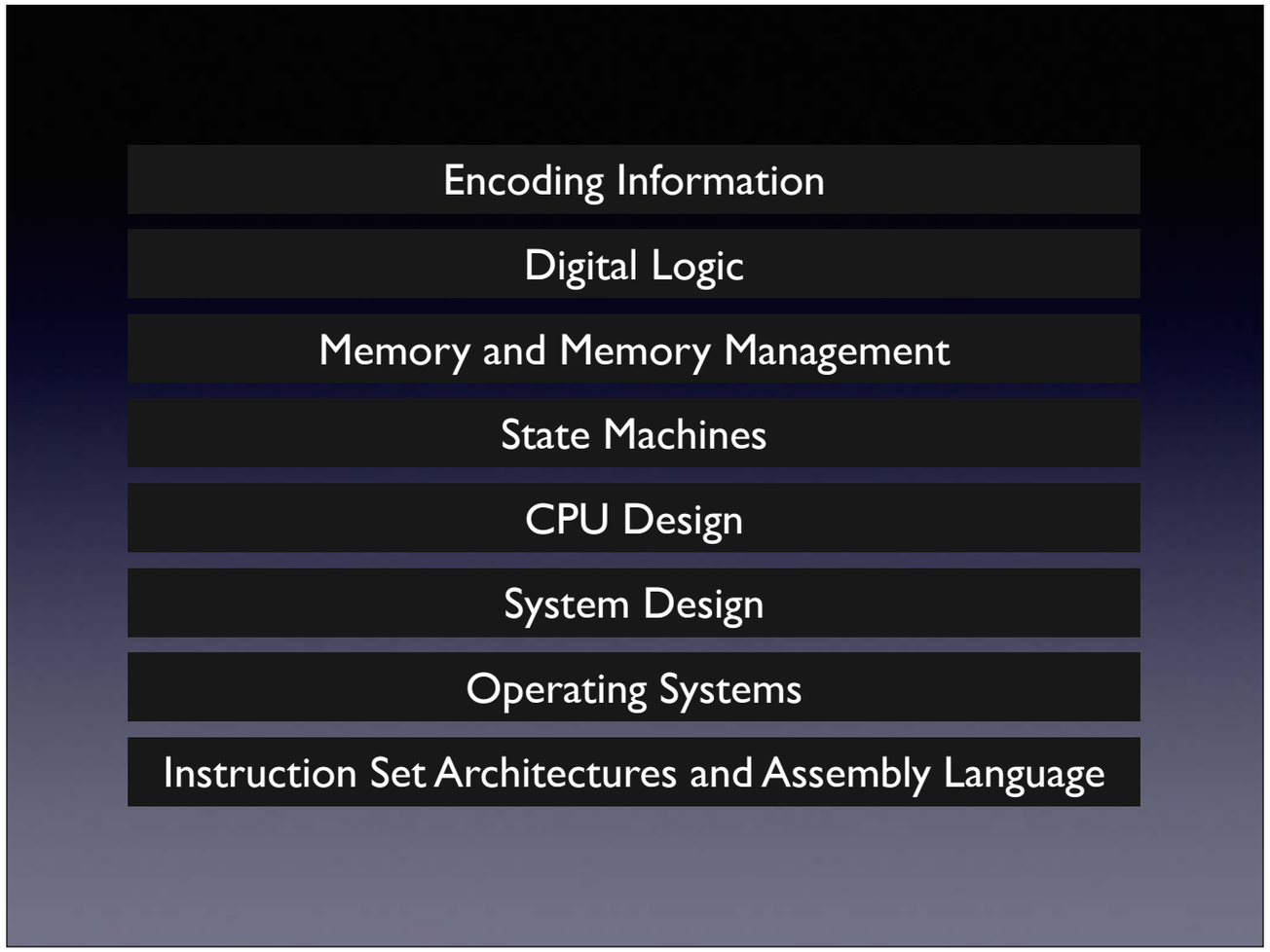
Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course



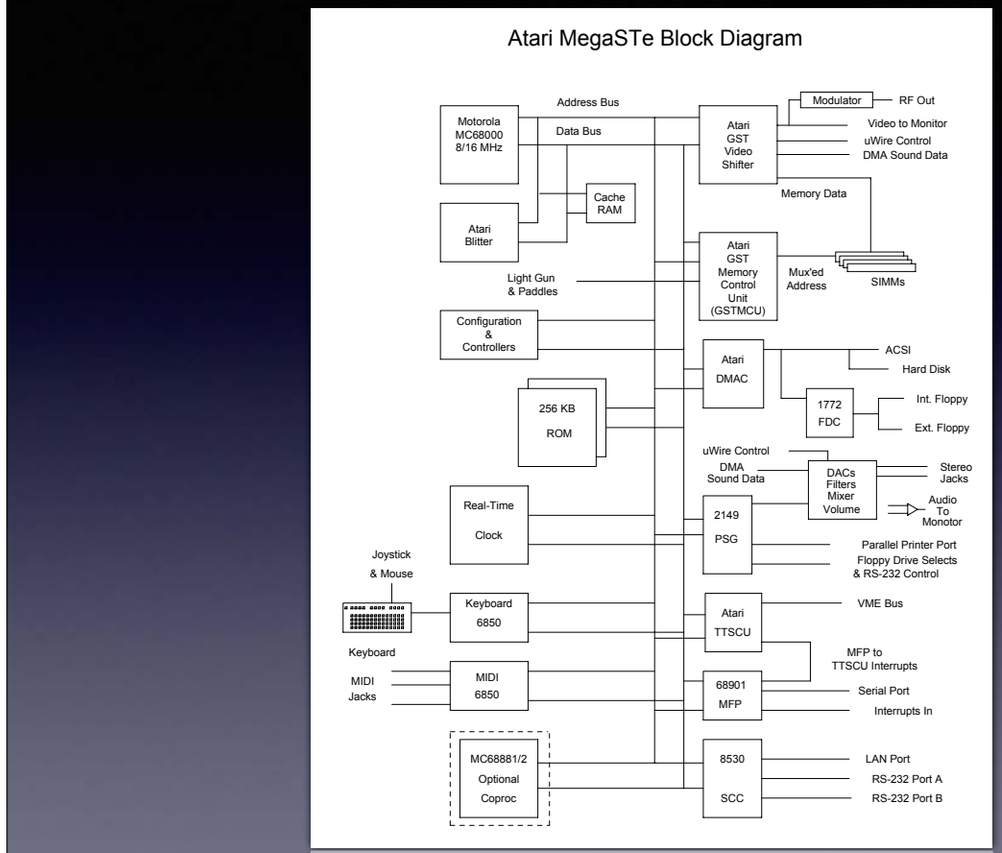
Other modules will pick up on some of these things in more detail
Spend a lot of time writing ARM code toward the end of the course

CSA Architecture

- Going to be based around two CPUs
  - ARM
  - Motorola 680x0 family
- Although primary focus will be ARM

What's inside a computer?

Take the ST to bits…

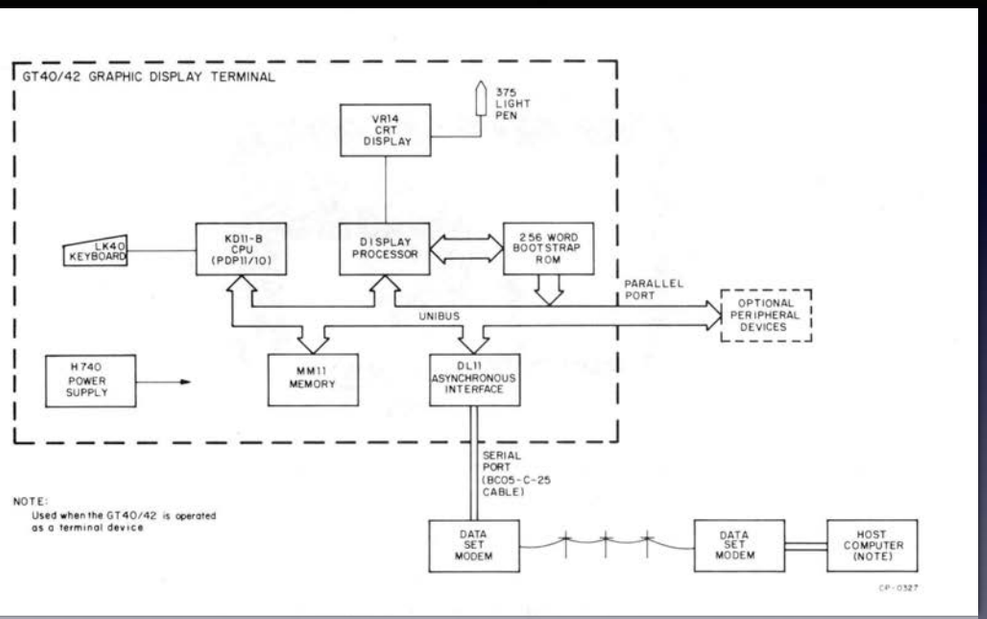


Block diagram of an Atari ST variant
Note how everything is connected by three buses— the address bus, data bus, Also a clock to keep everything in sync…

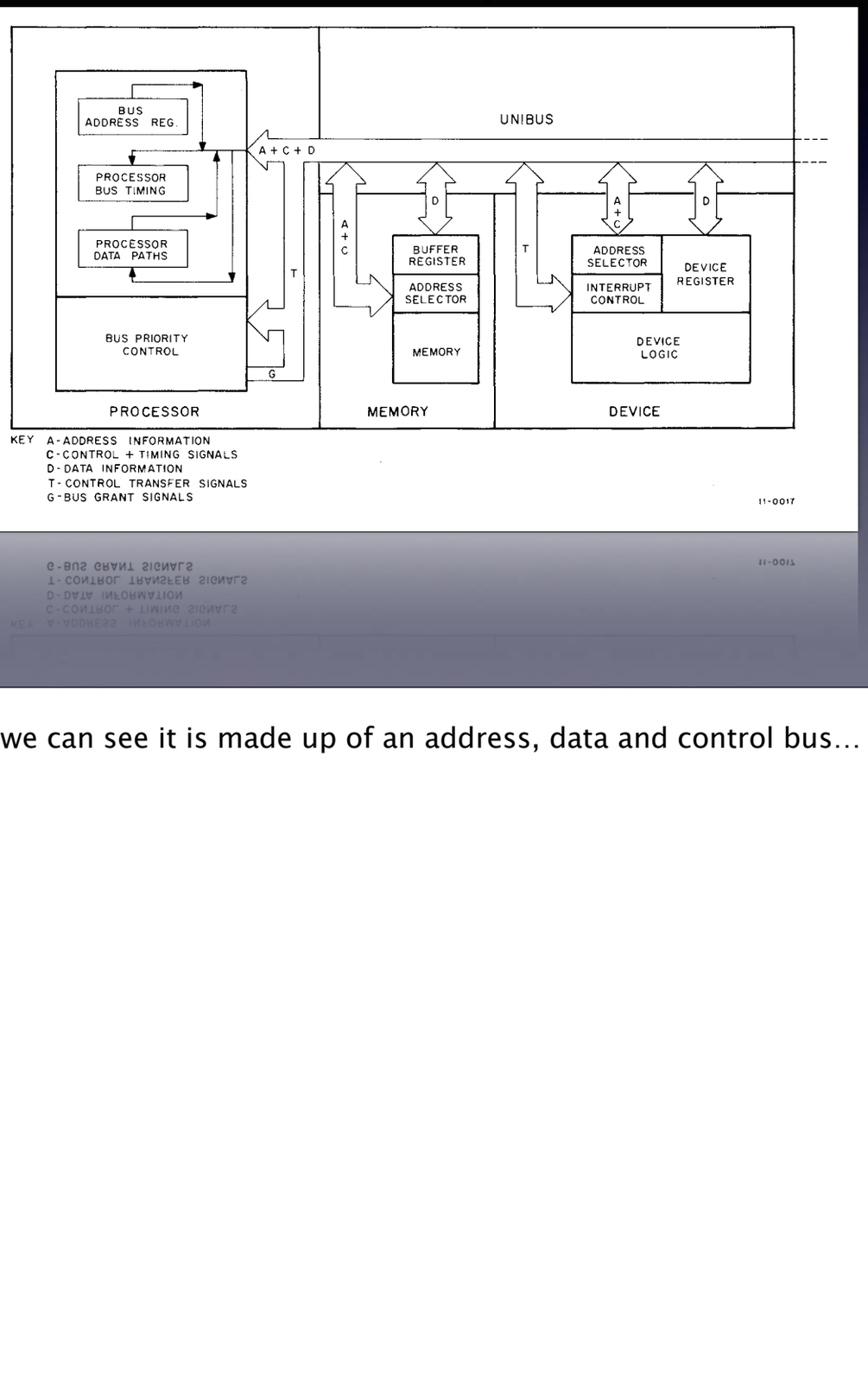
Buses

- Address bus — Location of what we wanting to access
- Data bus — the data we are accessing
- Control bus — signals saying what is happening

Are we reading or writing?
Has an interrupt occured
Common across pretty much all computer systems



DEC GT40 has a ‘unibus’ but if we look deeper we can see it is made up of an address, data and control bus…



DEC GT40 has a ‘unibus’ but if we look deeper we can see it is made up of an address, data and control bus…

Buses

- All parts of the system communicate through the buses
- Set the address lines to specify what they want
- Read or write the data from the data lines
- Set the control bits to specify what is happening

Are we reading or writing etc.

Buses

- Works very much like a courier
- He has an address which he goes to
  - To collect a parcel
  - Or deliver a parcel

Buses

- Unlike a road though, only one thing can use a bus at a time
- Otherwise the signals will interfere
- So there needs to be some pins to arbitrate who has control of the bus
- In a simple system, just the CPU has access but for real systems it gets more complex

But DMA (Direct Memory Access) etc can mean that other things want access…

Buses

- Just as with a courier, it takes time for the signals to travel down the bus
- Electrical signals will travel at the speed of light (roughly)
- Which gives us an interesting issue…

Speed of light

- Computers are synchronized to a clock
- Everything happens on the tick of the clock
- There's a period of time between each tick
- How far can the signals travel in that time?
- Can calculate it…

Speed of light

$d = st$

s is speed
t is time

or 10cm

Speed of light

$$d = st$$

$$t = \frac{1}{clock\ freq}$$

s is speed
t is time

or 10cm

Speed of light

$d = st$

$$t = \frac{1}{clock\ freq}$$

$$t = \frac{1}{3\text{x}10^9}\ \text{s}$$

s is speed
t is time

or 10cm

Speed of light

$d = st$

$$t = \frac{1}{clock\ freq}$$

$$t = \frac{1}{3\text{x}10^9}\ \text{s}$$

$$s = 3\text{x}10^8\ \text{ms}^{-1}$$

s is speed
t is time

or 10cm

Speed of light

$d = st$

$d = \frac{3 \times 10^8}{3 \times 10^9}$

$t = \frac{1}{clock\ freq}$

$t = \frac{1}{3 \times 10^9}$ s

$s = 3 \times 10^8$ ms^{-1}

s is speed
t is time

or 10cm

Speed of light

$d = st$

$$d = \frac{\cancel{3} \times 10^8}{\cancel{3} \times 10^9}$$

$$t = \frac{1}{clock\ freq}$$

$$t = \frac{1}{3 \times 10^9} \text{ s}$$

$s = 3 \times 10^8 \text{ ms}^{-1}$

s is speed
t is time

or 10cm

Speed of light

$d = st$

$$d = \frac{3 \times 10^{8}}{3 \times 10^{9}}$$

$d = 10^{-1}$ m

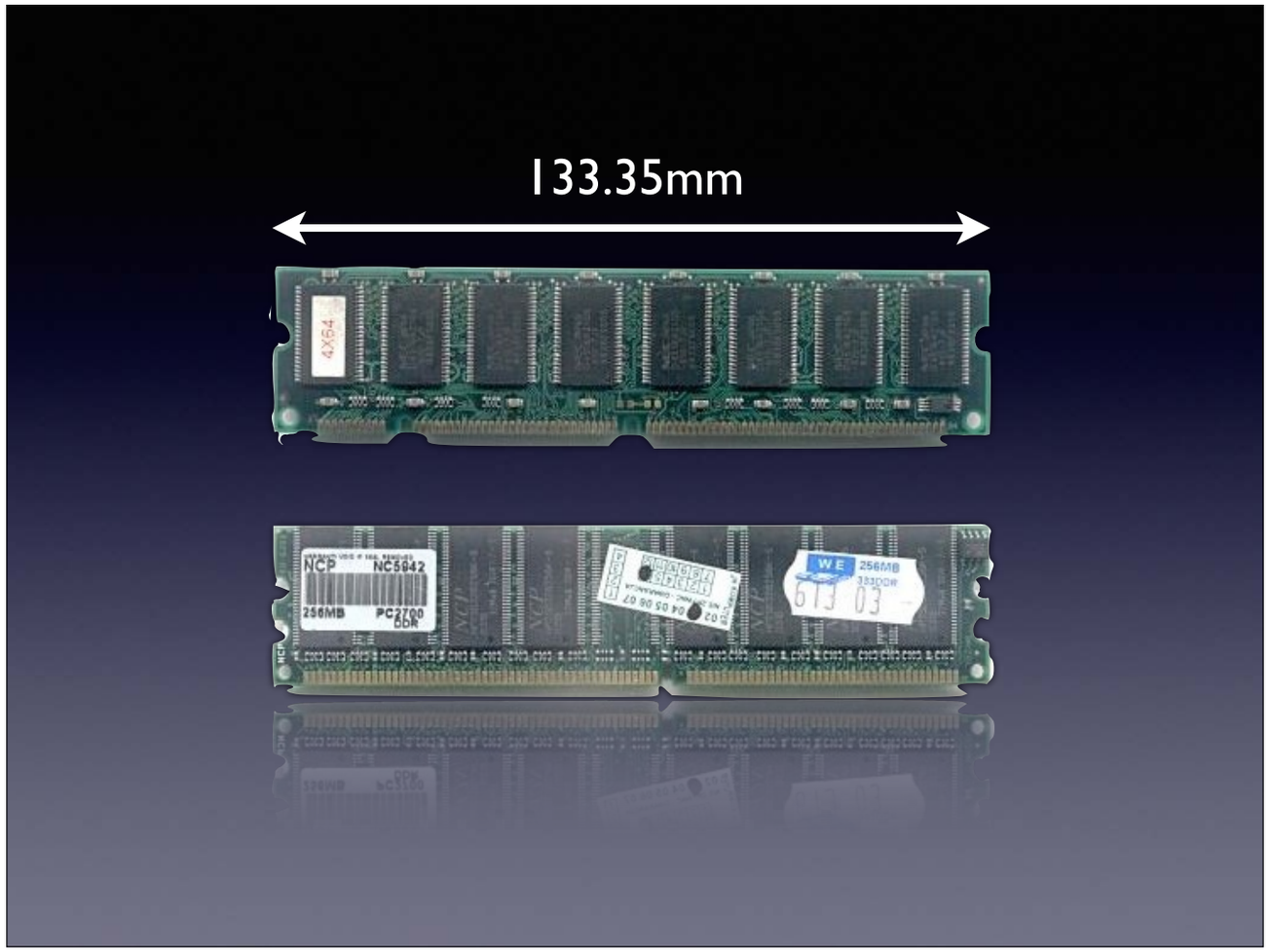
$$t = \frac{1}{clock\ freq}$$

$$t = \frac{1}{3 \times 10^{9}} \text{ s}$$

$s = 3 \times 10^{8}$ ms^{-1}

s is speed
t is time

or 10cm



Oops, takes longer than one clock cycle to get from one of a DIMM to another!

Leaky abstractions

- In reality, memory doesn't work that fast anyway
- But shows how things can leak through the abstractions
- While we think it might work one way, the implementation can be different
- Hidden behind the abstraction

Digital

- Only two states for the signal…
- Either on or off
- Take anything above a certain threshold to mean on
- And anything below to mean off

On and Off, 1 or 0, High or low, Asserted or not-asserted, true or false
All used to mean the same thing (in different contexts)

Digital

- Classic TTL signal levels (based on 5V system)
- Anything above 2V is on
- Anything below 0.8V is off
- There's a no-man's land between the two which is undefined…

Digital

- But how do we represent a number with just two states?
- Answer — binary numbers…

Numbers

- Various symbols have been used over the ages to represent numbers
  - Roman numerals
  - Arabic numbers (as we use today)
- All are an encoding for the underlying quantity

Numbers

- Based around ten because we have ten fingers
- But we also have counting systems using other bases
  - Time, for example…
- Computers just use a different encoding using two symbols

Because we only have two states,on or of, 0 and 1, high or low. True or false.

Decimal counting

Decimal counting

Decimal counting

8	8 Thousands
1	1 hundred
9	9 Tens
2	2

Decimal counting

8	8 Thousands	8000
1	1 hundred	100
9	9 Tens	90
2	2	2
		8192

Decimal counting

8	8 Thousands	8000	10
1	1 hundred	100	10
9	9 Tens	90	10
2	2	2	10

8192

Binary Counting

- Works the same as decimal counting
- Except we use powers of 2 rather than 10
- So we have units, 2s, 4s, 8s, 16s etc…

Decimal counting

Easy to convert from binary to decimal -- just add up which powers of two have

Decimal counting

Easy to convert from binary to decimal -- just add up which powers of two have

Decimal counting

1	1 Eight
0	0 Fours
1	1 Two
1	1 unit

Easy to convert from binary to decimal -- just add up which powers of two have

Decimal counting

1	1 Eight	8
0	0 Fours	0
1	1 Two	2
1	1 unit	1
		11

Easy to convert from binary to decimal -- just add up which powers of two have

Decimal counting

1	1 Eight	8	2
0	0 Fours	0	2
1	1 Two	2	2
1	1 unit	1	2
		11	

Easy to convert from binary to decimal -- just add up which powers of two have

Decimal to Binary

- Several ways to convert from decimal into binary
- Could just test each power of two against the number and see if the number is bigger (if it is then subtract that number from it and repeat)
- Or we can do it by repeated division…

Go demo on the white board…

Binary Numbers

- Often written out with leading zeros
- Up to a certain number of bits — usually a multiple of eight
- So 42 is written as 00101010, to 101010
- Might also see it written as 0b00101010 to signify it is binary…

bits = binary digit

Octal and Hexadecimal

- Binary numbers get long and unwieldy
- Try scribbling 64bits down on a piece of paper…
- Tend to use octal or hexadecimal number systems instead for ease

Octal and Hexadecimal

- Octal is a base-8 system
- Hexadecimal is base-16
- Crucially, both of these are powers of two
- So each octal digit equates to three bits
- Each hex digit equates to four bits very easy to convert

Go show how it works on the white board

Binary	Hexadecimal	Octal	Decimal
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	8	10	8
1001	9	11	9
1010	A	12	10
1011	B	13	11
1100	C	14	12
1101	D	15	13
1110	E	16	14
1111	F	17	15

Hex binary octal table
Note how hex uses letters to signify numbers greater than 9

Binary to Octal (or Hex)

- Start from the right hand side
- Group together three (or four for hex) bits
- Convert to octal or hex digit
- Repeat for the next three (four) bits
- And so on…
- And vice versa…

This approach doesn’t work for decimal